Extracted from:

Exploring Graphs with Elixir

Connect Data with Native Graph Libraries and Graph Databases

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Your Elixir Source

Exploring Graphs with Elixir

Connect Data with Native Graph Libraries and Graph Databases

Tony Hammond

Series editor: *Bruce A. Tate*
Development editor: *Jacquelyn Carter*

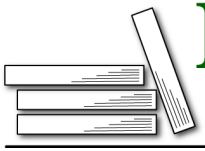
Exploring Graphs with Elixir

Connect Data with Native Graph Libraries and Graph Databases

Tony Hammond

The Pragmatic Bookshelf

Raleigh, North Carolina



For our complete catalog of hands-on, practical, and Pragmatic content for software developers, please visit *https://pragprog.com*.

The team that produced this book includes:

CEO: Dave Rankin
COO: Janet Furlow
Managing Editor: Tammy Coron
Series Editor: Bruce A. Tate
Development Editor: Jacquelyn Carter
Copy Editor: Corina Lebegioara
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics
Founders: Andy Hunt and Dave Thomas

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-840-6
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—November 2022

Introduction

This is a book about connections and disconnections. It's about graph data structures and Elixir. It's about distributed data and distributed compute. It's about the graph.

Graphs as an organizational pattern operate at all levels, from the smallest static structures—for example, chemical compounds such as a water molecule—to large-scale dynamic structures—for example, web-based social networks such as Facebook or Twitter. They allow us to link data points together with a minimum of fuss or overhead. As often noted, graphs lend themselves well to a schema-less style of data connection.

What kinds of graph models are there and how can we access them from our preferred language, Elixir? This is what we'll look to answer in this book. How easy is it to get started with graphs? How can we build native graph structures in Elixir? How can we query graph databases with Cypher, Gremlin, and GraphQL? How can we query linked open data with SPARQL?

We'll see how.

Graph structures are great at helping us connect data, but in practice, graphs tend to be disconnected from each other both physically in terms of location and conceptually in terms of a data model. We want to be able to move our graph data between graph models and graph databases. We want to straddle across different implementations.

We'll aim to get a better understanding of graph models and of Elixir packages that can be used to build graphs and query graph databases. We'll also see how we can use Elixir's support for concurrency in managing connected and distributed data.

Who This Book Is For

This book is directed primarily at the practicing Elixir programmer who wants to get a better understanding of the graph-processing landscape and see both

what support already exists within Elixir and also what can be readily accessed from Elixir.

We'll be mainly interested in using graphs for creating and supporting information networks. The aim is more to move data around and work with, and across, different types of graph models rather than to explore graph algorithms per se. There are other resources for that. Here we want to focus on the graph models themselves and how we might bridge from one model to another.

The book works its way through developing a complete application and as such can be used as an Elixir learning aid. Some basic knowledge of Elixir is assumed, although the book proceeds at a gentle pace and will explain new concepts as they are encountered. The book will use a number of standard Elixir technologies:

- umbrella apps
- behaviours
- macros
- protocols
- OTP

The book may also be of interest to those who don't know Elixir but have some acquaintance with graphs and are curious to learn more about how the two main graph paradigms—property graphs and RDF graphs—intersect and the tooling to make them interoperate. In this case, the use of Elixir code examples throughout the book may be read as a kind of pseudocode.

How to Read This Book

The book is organized into three parts:

- Part I—Graphs Everywhere
- Part II—Getting to Grips with Graphs
- Part III—Graph to Graph

Part I will give a simple introduction to graphs and networks and then go on to set up the project that we will develop together. This entails creating an umbrella app and a common library app that will be used by the apps developed later.

We'll also spend a little time setting up project storage areas for graphs and queries for the various graph types and data structures for managing access to those areas. The goal of this exercise is to make it much simpler to reuse these components so that we can focus on graph manipulations rather than file operations.

Part II will be a hands-on investigation of a number of Elixir packages for working with graphs:

- libgraph
- bolt_sips
- rdf, sparql, sparql_query
- gremlex
- dlex

We'll first look at developing native graph models within Elixir and then look at applied graph models and interacting with graph databases. In this area, we'll look at the two main paradigms for modeling information networks: property graphs and RDF graphs.

We'll work with various graph types and their associated graph query languages and will introduce shorthand query forms to make querying simpler.

Part III will then look at moving between graphs, how we can transform graph models from one standard form to another, and how we can use Elixir's concurrent processing capabilities to manage distributed graphs and distributed compute.

That at least is the plan.

About the Code

The following conventions are followed in this book.

If something needs to be invoked on the command-line terminal, you'll see a $ sign preceding it in the text:

```
$ iex
Erlang/OTP 24 [erts-12.3.1] ...

Interactive Elixir (1.13.4) - press Ctrl+C to exit (type h() ENTER for help)
iex>
```

If something needs to be invoked from within the Elixir interactive shell IEx, you'll see a iex> sign preceding it in the text:

```
iex> g = Graph.new |> Graph.add_vertices([:a, :b])
```

Otherwise, if it's a source file listing, you'll usually see a file name banner:

apps/native_graph/lib/native_graph/builder.ex

```
def random_graph(limit) do
  for(n <- 1..limit, m <- (n + 1)..limit, do: do_evaluate(n, m))
  |> Enum.reject(&is_nil/1)
  |> Enum.reduce(
    Graph.new(),
```

```
    fn [rs, re], g ->
      Graph.add_edge(g, rs, re)
    end
  )
end
```

Some other conventions are also used.

While we follow the general convention to use pipes only for functions with multiple arguments, we do make an exception for query strings. Given the length of some of the query strings we will encounter, it seems more natural to pipe the query string into the query function instead of adding it as an argument:

```
iex> "MATCH p = ()-[*]->() RETURN DISTINCT length(p)" |> cypher!
[%{"length(p)" => 1}, %{"length(p)" => 2}, %{"length(p)" => 3}]
```

This also allows us to focus on the query itself and not the query function:

```
iex> """
...> {
...>   q(func: has(EX)) {
...>     uid
...>     EX { uid }
...>   }
...> }
...> """ |> dgraph!
  %{"q" => [%{"EX" => [%{"uid" => "0x4ebb"}], "uid" => "0x4eba"}]}
```

The emphasis in this book will mostly be on interactive exploration using IEx so certain practices are followed to simplify data entry and the number of keystrokes. For example, importing functions will be common so that we can reuse the same functions for different graph models, such as `query_graph`, `read_graph`, and others.

About the Software

To follow along with the book, you should have Elixir 1.10+ installed. The book will guide you through setting up an umbrella application for a graph test bed using a variety of graph databases. Instructions for installing the graph databases are given in an appendix.

This book was written in 2022, and the versions of the various software packages used in this book's development are listed in the table shown on page ix.

Software	Version Tested
Languages	
Elixir	1.13.4
Databases	
Dgraph	21.12.0
GraphDB Free	9.11.1
Gremlin Server	3.6.0
Neo4j Community Edition	4.4.5

Online Resources

The sample code for this book is available from the book page on the Pragmatic Programmers website.[1] You'll also find the errata-submission form there, where you can report problems with the text or make suggestions for future versions.

Tony Hammond
November 2022

1. https://pragprog.com/titles/thgraphs